DOE P3HPC FORUM 2020

Argonne NATIONAL LABORATORY

PREPARING PERFORMANCE PORTABLE QMCPACK FOR EXASCALE

QMCPACK

YE LUO
Computational Science Division
& Leadership computing facility
Argonne National Laboratory

PETER DOAK
Oak Ridge National Laboratory

PAUL KENT
Oak Ridge National Laboratory

U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

Sep 1st 2020. Virtual

ACKNOWLEDGEMENTS

- Thanks to
  - QMCPACK developer team
  - Johannes Doerfert (ANL)
  - Shilei Tian (Stony Brook University)
  - Oscar Hernandez (ORNL)

Supported by:
This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

Computational resource:
We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

PERFORMANCE PORTABLE DESIGN

QMCPACK

In a nutshell

- QMCPACK, is a modern high-performance open-source Quantum Monte Carlo (QMC) simulation code for electronic structure calculations of molecular, quasi-2D and solid-state systems.
- The code is C/C++ and adopts MPI+X(OpenMP/CUDA)
- Monte Carlo: massive Markov chains (walkers) evolving in parallel. 1st level concurrency.
- Quantum: The computation in each walker can be heavy when solving many body systems (electrons). 2nd level concurrency.

MAPPING CONCURRENCY TO PARALLELISM

Monte Carlo can be a challenge for parallelism

Friendly
Unfriendly

- Walkers N_w are not data parallel but task parallel
  - Workload per electron move depends on accept/reject. GPU
  - Workload per step moving all the electrons is roughly equal. CPU
- Electrons are data parallel
  - Kernels are $O(N_e^{2-3})$ per sample. Large N_e CPU. Small N_e GPU.
  - Naturally, N_e vector computation utilizing SIMD and SIMT. CPU/GPU
- Simulations need N_e from 10 to 10000 depending on the scientific questions
  - Use N_w and N_e to balance compute node efficiency and time-to-solution.
  - Need a tailored approach for performance portability beyond programming models.

MAPPING CONCURRENCY TO PARALLELISM

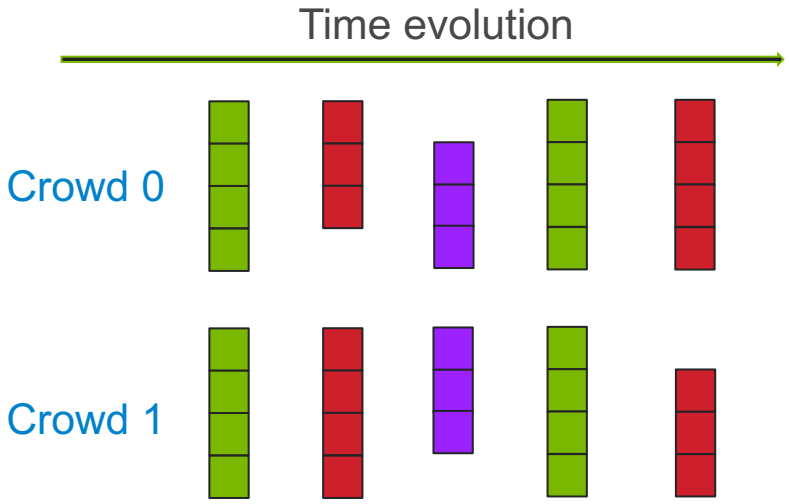
Need a flexible scheme at high level for all sizes of Nw and Ne

- Our CPU/GPU portability experience since 2010
  - Walker batching saves GPU kernel overhead in small problems.
  - Lock-step algorithm has performance penalty with large problem sizes.
  - Incompatible internal APIs and diverged code paths without fallback for missing features.
    - CPU QMC drivers have no walker batching
    - Legacy CUDA QMC drivers are very bad with large problem sizes
- Requirement for performance portable code
  - Feature complete
  - Computationally intensive pieces accelerated and selected at run time
  - Single source is desired but architectural specialization is possible and only allowed at the bottom level.
  - Not restricted to a particular programming model at high abstraction level

MAPPING CONCURRENCY TO PARALLELISM

Design unified QMC driver design for flexible dispatching

- The walker population with a node is subdivided into crowds.
  - Legacy CPU drivers have crowd size 1.
  - Legacy CUDA drivers have 1 crowd.
- Walkers within a crowd evolve in lock step at every single electron move. Data parallelism.
- Walkers between crowds are not synchronized until all the single electron moves are completed within a step. Task parallelism.
- Lower levels have both batched and non-batched APIs. Fallback is by default and can be specialized.



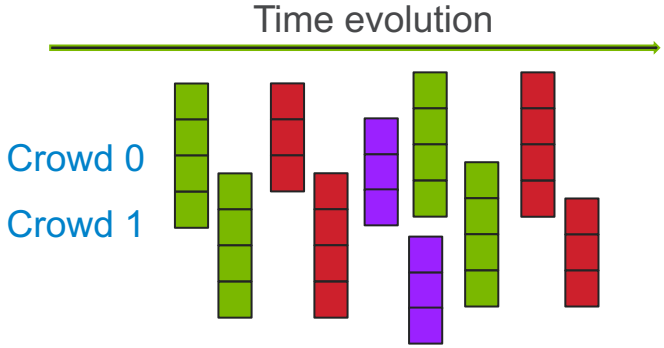
Unified batched QMC driver design

PERFORMANCE PORTABLE IMPLEMENTATION

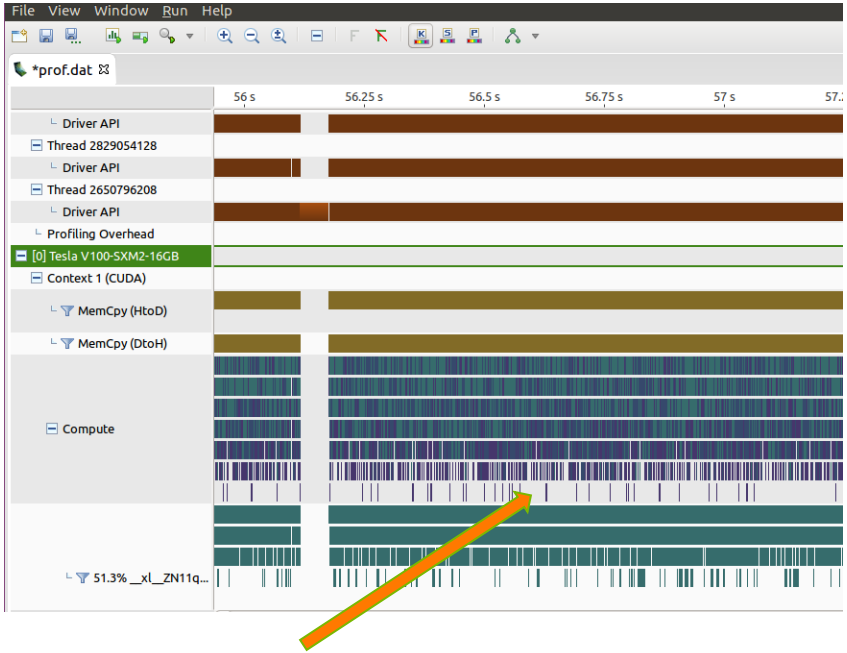
MAPPING CONCURRENCY TO PARALLELISM

Threads and streams

- Crowds are mapped to CPU threads.
  - No idle. Nested threads are optional.
- Crowds leverage GPU streams/queues explicitly or implicitly.
- Desynchronized crowds keep the computing device busy.



Unified batched QMC driver execution on an accelerator

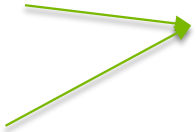


MiniQMC concurrent crowds
IBM XL OpenMP runtime

IMPLEMENTATION STRATEGY

The current status

- Spline single particle orbital evaluations are implemented using OpenMP target offload
- Slater determinant updates are implemented using cuBLAS/cuSolver.
- Both batched and non-batched code path are specialized for maximal performance.
- Non-local pseudopotential evaluation supports additional batching for quadrature points evaluation.
- Jastrow factors and distance tables remains on the CPU for the moment.



Majority cost

Complicated algorithm but heavy cost.

Complicated algorithm but light cost.

PERFORMANCE PORTABILITY WITH OPENMP

A touch journey in 2019

- 2019 PPP meeting, IBM XL C/C++ compiler is the only working compiler for QMCPACK
- 2019 Dec 2nd. https://github.com/QMCPACK/miniqmc/wiki/OpenMP-offload

Compiler	Clang 9	AOMP 0.7-4	XL 16.1.1-3	Cray 9.0	GCC 9.2
device	NV	AMD	NV	NV	NV
math header conflict	F	P	P	P	P
math linker error	P	P	P	P	P
declare target static data	P	P	P	P	F
static linking	F	P	P	P	F
check_spo	FR	FW	P	P	FL
check_spo_batched	FR	P	P	P	FL
miniqmc_sync_move	FR	P	P	P	FL

Cray 9.1 inherits Clang 9 math function issues.

PERFORMANCE PORTABILITY WITH OPENMP

A lot of exciting improvements in 2020

- 2020 Aug 30th. https://github.com/QMCPACK/miniqmc/wiki/OpenMP-offload

Compiler	Clang 11	AOMP 11.8-0	XL 16.1.1-5	OneAPI beta08	Cray 9.0	GCC 10.2
device	NVIDIA	AMD	NVIDIA	Intel	NVIDIA	NVIDIA
math header conflict	Pass	Pass	Pass	Pass	Pass	Pass
complex arithmetic	Pass	Pass	Pass	Pass	Fail	-
declare target static data	Pass	Pass	Pass	-	Pass	Fail
static linking	Fail	Pass	Pass	Pass	Pass	-
multiple stream	Pass	Pass	Pass	Functioning	Functioning	-
check_spo	Pass	Pass	Pass	Pass	Pass	-
check_spo_batched	Pass	Pass	Pass	Pass	Pass	-
miniqmc_sync_move	Pass	Pass	Pass	Pass	Pass	-

Workaround in cmake

INTERACT WITH COMPILER DEVELOPERS

QMCPACK OpenMP offload works cross platforms

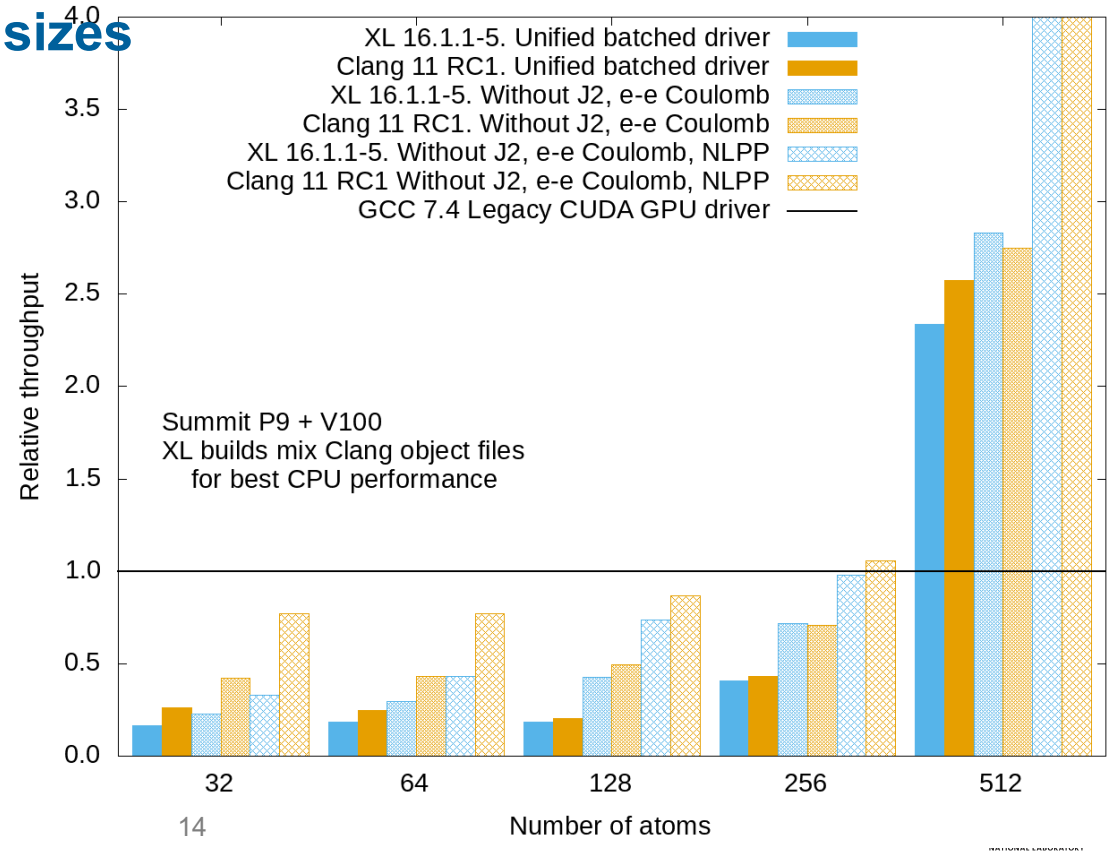
- LLVM and SOLLVE fixed 17/20 bug reports. 4 requested optimization added.
- AOMP fixed 9/14.
- Contribute tests to vendor compiler team via early hardware access program.
- Having our own testing. https://cdash.qmcpack.org

Ye-Ryzen-Box	AOMP-Offload-Complex-Mixed-Release	0	1	227	16 hours ago
Ye-Ryzen-Box	AOMP-Offload-Real-Mixed-Release	0	1	257	17 hours ago
Ye-Ryzen-Box	AOMP-Offload-Complex-Release	0	1_{-211}	227^{+211}	18 hours ago
Ye-Dell-Laptop	OneAPI-Offload-Real-Mixed-Release	0	2	253	18 hours ago
Ye-Dell-Laptop	OneAPI-Offload-Real-Release	0	1	277	18 hours ago
bora.alcf.anl.gov	ClangDev-Offload-CUDA-Complex-Mixed-Release	0	1_{-2}	227^{+2}	
bora.alcf.anl.gov	ClangDev-Offload-CUDA-Complex-Release	0	2	226	
bora.alcf.anl.gov	ClangDev-Offload-CUDA-Real-Mixed-Release	0	0	258	
bora.alcf.anl.gov	ClangDev-Offload-CUDA-Real-Release	0	0	281	

PERFORMANCE ON SUMMIT

NiO benchmark at various sizes

- Clang becomes better than XL in overall performance.
- Clang has a more efficient runtime but slower kernels.
- At 512 atom size. New GPU code is way more efficient.
- With optimized code paths, the new GPU implementation reaches at least 50% performance of legacy CUDA code which offloads more to GPU and use async computation.



IMPROVEMENTS NEEDED IN QMCPACK

Keep effort in making the code better

- QMCPACK developers put a large effort on refactoring the existing code and adding a better design. The progress is not easily visible to the outside but fundamentally important to make all things happen. Will keep doing this non-stop.
- Need to further reduce data movement and synchronization. This requires making more computation go asynchronously.
- Use algorithmic innovation to fundamentally solve problems.

IMPROVEMENTS NEEDED OUTSIDE QMCPACK

Software stack missing pieces

- In OpenMP, we need
  - target nowait async support with task dependency.
  - More GPU related 5.0 features implemented.
  - Interoperability with vendor programming models.
  - Vendor compilers more reliable and capable.

- Libraries
  - Need batched BLAS1/2, see online manual and cublas_missing_functions

- Tools
  - OpenMP friendly debugger and profiler.